Applied Mathematics and Computation 232 (2014) 292–302

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Minesweeper strategy for one mine

Shahar Golan

Department of Computer Science, Ben-Gurion University of the Negev, POB 653, Beer-Sheva 84105, Israel

ARTICLE INFO

Keywords:
Graph theory
Strategy
Probability
Algorithm
Games
Minesweeper

ABSTRACT

Minesweeper is a popular single player game. Various strategies have been suggested in order to improve the probability of winning the game. In this paper, we present an optimal strategy for playing Minesweeper on a graph when it is known that exactly one cell is mined.

1. Introduction

Minesweeper is a popular single player game, distributed with the Microsoft Windows operating system. It consists of a grid of covered cells (see Fig. 1), some of which are *mined cells*, i.e., contain mines. The cells not containing mines are *free cells*. The mined cells are randomly and uniformly distributed in the grid. Each free cell contains information regarding the number of its mined neighbors (where cells adjacent diagonally are also considered as neighbors). At each move, the player may uncover a cell. If the cell is mined, the game is lost; if it is free, the information contained in it is revealed, and the player may use it. The goal of the game is to uncover all free cells, leaving all mined cells covered (see Fig. 2). The total number of mines is usually given as well.

Various approaches were examined for planning a strategy for maximizing the probability of winning Minesweeper. In [1], software is provided, enabling one to add his strategy (as a Java program) and test its performance on a predefined benchmark. This benchmark consists of three grids: Beginner – an 8×8 grid with 10 mines, Intermediate – 15×13 with 40 mines, and Expert – 30×16 with 99 mines, where the mined cells are chosen in each case uniformly at random out of all possibilities. Several basic strategies are also provided in [1]. One of these regards the problem as a constraint satisfaction problem. A more sophisticated version of this strategy (devised independently) was examined in [2]. The first strategy was found to achieve winning rates of 71%, 36% and 26%, respectively. The second achieved winning rates of 80%, 44% and 34%, respectively for the three grids above. The statistics were taken in a setting where the first move is always safe (i.e., the mines are distributed between all cells except for the one selected by the player at the first move). In [3–5], genetic algorithms were used in order to solve the problem. A learning agent was introduced in [6]. The results of these AI attempts were no better than the human tailored algorithms. There have been also some attempts to use neural networks [7]. Another approach was presented in [8], where the grid is represented by a cellular automaton.

The configuration given to the player before each move, consisting of a grid with some uncovered cells containing known information will be referred to as a *Minesweeper grid* (henceforth MS grid). In such a grid there might be covered cells that can be positively inferred as free. Such cells are *safe*. A rational player may be assumed to always try to find and open a safe cell before guessing. An *assignment* $\mathcal{A}$ is a function from the Minesweeper grid's cells to the set {free, mined}. A *legal assignment* is an assignment such that all uncovered cells are assigned free and contain the correct values (i.e., exactly

E-mail address: golansha@gmail.com

http://dx.doi.org/10.1016/j.amc.2014.01.045
0096-3003/

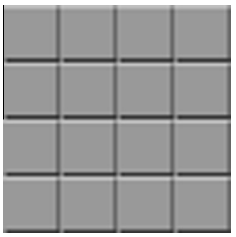
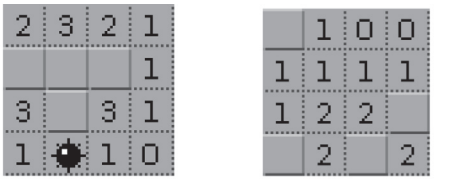


Fig. 1. The initial state.



(a) A lost game (b) A winning state

Fig. 2. Ending states of Minesweeper

the number of mined neighbors they have). A covered cell is safe if and only if all legal assignments give it the value 'free'. Thus, an important part of any strategy is determining whether there exists a legal assignment where a given cell is mined.

Several variations of the Minesweeper game were defined and analyzed. One of these is to discard the requirement that the board must be a grid, and consider Minesweeper on general graphs. An *MS graph* is the analogue of an MS grid for graph. Thus, in an MS graphs, each free vertex contains the information regarding the number of its mined neighbors. A method for calculating the number of legal assignments for a partially uncovered MS graph is given in [9]. A new version of Minesweeper, where the board consists of triangle cells is presented in [10]. Infinite versions of Minesweeper are given in [11,12].

In this paper we design an optimal strategy for playing Minesweeper on general graphs, when it is known that exactly one cell is mined.

In Section 2 we present the main results. In Section 3 we introduce a concept called *guess component*. The proofs regarding Minesweeper problems with one mine are given in Section 4.

2. The main results

A *strategy* for minesweeper is an iterative algorithm, which determines which cells to uncover in the course of the game. In every iteration of the algorithm, a covered cell is chosen and uncovered. Thus, for any MS graph (not necessarily containing exactly one mined cell), any strategy can be described by the scheme given in Algorithm 1.

Algorithm 1. A generic scheme for a strategy

```
GenericStrategy (Graph G)
  Initialize auxiliary data structures
  Set Covered ← V(G)
  while |Covered| > TotalNumberOfMines
    v ← Choose(G, Covered)
    uncover v
    if label(v) = *
      print lost
      return
    Update auxiliary data structures
    Covered ← Covered \ {v}
  print won
```

For a strategy s and a graph G, denote by $P_s(G)$ the probability of finding all the mines in G without uncovering any of their cells when using s. An *optimal strategy* s^* is a strategy such that $P_{s^*}(G) \geqslant P_s(G)$ for every strategy s. Denote $P^*(G) = P_{s^*}(G)$. In this section we present an optimal strategy for playing Minesweeper on an MS graph known to include exactly one mined cell.

Without loss of generality we may assume that an optimal strategy will always open safe cells before trying to guess. The problem of finding the set of safe cells in an MS graph is known to be NP-complete in general [13], and even for graphs with bounded vertex degrees $d < 3$ [14]. However, in the case of an MS graph with one mined cell, this problem can be easily solved. Throughout the rest of this section, $G = (V, E)$ will denote an MS graph with a single mine. Algorithm 2 (in which the method OptimalGuess is not specified) is a scheme for an optimal strategy, taking into account the above considerations.

Algorithm 2. A scheme for an optimal strategy

```
GenericOptimalStrategy (Graph G)
  Set Safe ← ∅
  Set Covered ← V
  while |Covered| > 1
    v ←OptimalChoice(G, Covered, Safe)
    uncover v
    if label(v) = *
      print lost
      return
    Safe ← Safe \ {v}
    Covered ← Covered \ {v}
    Safe ← SafeCells (G, Covered, Safe, v)
  print won
OptimalChoice (Graph G, Set Covered, Set Safe)
  if Safe = ∅
    return OptimalGuess (G, Covered)
  else
    return an arbitrary cell in Safe
SafeCells (Graph G, Set Covered, Set Safe, cell v)
  if label(v) = 0
    return (Safe ∪ N(v)) ∩ Covered
  else
return (Safe ∪ (V \ N(v))) ∩ Covered
```

In this scheme, the set *Safe* is the list of covered cells that can be deduced to be free on the basis of the information gained from the uncovered cells. $N(v)$ is the set of neighbors of a vertex $v \in V$.

Denote by time i the "time" when the ith iteration of the **while** loop is started. Denote by $Safe_i$ and $Covered_i$ the sets *Safe* and *Covered*, respectively, at time i.

The following lemmas prove the correctness of Algorithm 2 and find its performance in terms of space and time.

Lemma 2.1. *For every $i \geqslant 1$, at time i:*

1. $Safe_i$ is free of mines.
2. For every cell v in $Covered_i \setminus Safe_i$, there exists a legal assignment for G, in which v is mined.

We have thus reduced the specification of the strategy to the definition of the method OptimalGuess.

Lemma 2.2. *If $Safe_i = \emptyset$, then every two cells in $Covered_i$ have the same set of uncovered neighbors. Additionally, all cells in $Covered_i$ have the same probability of being the mined cell.*

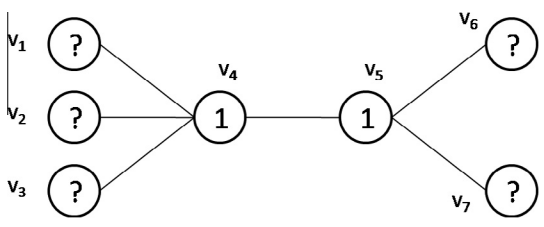


Fig. 3. A counter-example to Lemma 2.2 when G contains two mines.

Note that the lemma does not hold in general when there is more than one mine. Indeed, for the graph in Fig. 3, for example, the probability of v_1 being mined is $\frac{1}{3}$, while that of v_6 is $\frac{1}{2}$.

For a set of vertices $H \subseteq V$, denote by $P^*(H)$ the winning probability of the subgraph of G induced by the set H.

Corollary 2.3. *If by time i the game is not yet over and $Safe_i = \emptyset$, then the probability of winning by continuing with the original optimal strategy is $P^*(Covered_i)$.*

Thus, we may assume that the method OptimalGuess is always applied to an MS graph with no uncovered vertices. In particular, to specify our strategy we only need to define OptimalGuess for such graphs. After the first guess is made, s^* chooses iteratively cells from *Safe* until it is depleted. If $Safe_i = \emptyset$ we may ignore all vertices in $V \setminus Covered_i$ and treat $Covered_i$ as a new MS graph. (Here and elsewhere we shall often not distinguish between a set of vertices and the graph induced by this set.)

We now present our main result:

Theorem 2.4. Algorithm 2, *when calling Algorithm 4 for the method OptimalGuess, is an optimal strategy. This algorithm takes $O(|V|^4)$ time and requires $O(|V|^2)$ memory.*

The following propositions give $P^*(G)$ for several families of graphs.

Proposition 2.5. *For an MS tree T, with n vertices and a set L of leaves:*

$$P^*(T) = \frac{\min(n-1, n-|L|+|N(L)|)}{n}.$$

An MS grid is a grid where cells diagonally adjacent are also considered neighbors.

Proposition 2.6. *For an $n_1 \times n_2$ MS grid G with $n_1 \leqslant n_2$*

$$P^*(G) = \begin{cases} \frac{n-1}{n}, & n_1 \neq 2, \\ \frac{1}{2}, & n_1 = 2, n_2 > 2, \\ \frac{1}{4}, & n_1 = n_2 = 2, \end{cases}$$

where $n = n_1 n_2 \geqslant 2$.

Denote by $G(n,p)$ the random graph on n vertices, is a graph with n vertices and in which every possible edge occurs with probability p, independently of other edges.

Proposition 2.7. *The probability that $P^*(G(n,p)) = \frac{n-1}{n}$ tends to 1 for any $0 < p < 1$ as n tends to infinity.*

3. Guess components

We recall that $N(v)$ is the set of neighbors of a vertex $v \in V$. A *guess component* of G is a set of vertices H of G, such that

$$N(v_1) \setminus H = N(v_2) \setminus H, \quad v_1, v_2 \in H.$$

Example 3.1. The guess components of the graph depicted in Fig. 4 are the subgraphs induced by the sets $\{v_1\}, \{v_2\}, \{v_3\}, \{v_4\}, \{v_5\}, \{v_6\}, \{v_1, v_3\}, \{v_1, v_2, v_3\}$ and $\{v_1, \ldots, v_6\}$. Note that the example demonstrates that an induced subgraph of a guess component is not necessarily a guess component.

The importance of this notion derives from the following immediate consequence of Lemma 2.2.

Corollary 3.2. *If $Safe_i = \emptyset$, then $Covered_i$ is a guess component.*

Lemma 3.3. *Let G be a graph containing two intersecting guess components H_1, H_2 with $H_1 \setminus H_2 \neq \emptyset$ and $H_2 \setminus H_1 \neq \emptyset$. Then:*

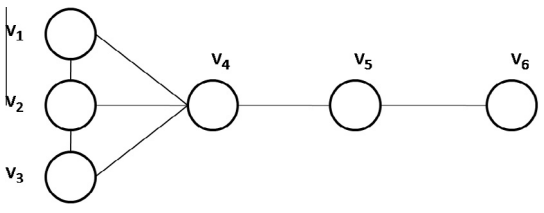


Fig. 4. An example of a guess component.

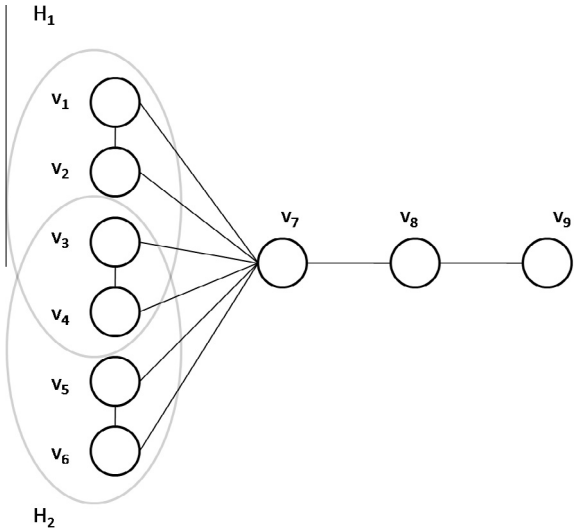


Fig. 5. An example of two intersecting guess components.

1. The set $[H_1 \times (H_2 \setminus H_1)] \cup [H_2 \times (H_1 \setminus H_2)]$ is either contained in $E(G)$ or disjoint from it.
2. The set $H_1 \cup H_2$ is a guess component.
3. The set $H_1 \cap H_2$ is a guess component.
4. The set $H_1 \setminus H_2$ is a guess component.
5. The set $H_1 \Delta H_2$ is a guess component.

Example 3.4. Consider the graph depicted in Fig. 5. The subgraphs H_1 and H_2 induced by the sets $\{v_1, v_2, v_3, v_4\}$ and $\{v_3, v_4, v_5, v_6\}$, respectively, satisfy the requirements of H_1 and H_2 in the last lemma.

Remark 3.5. The difference $H_1 \setminus H_2$ between two guess components (and even the complement of a guess component) is not necessarily a guess component when $H_1 \subsetneq H_2$.

Proof of Lemma 3.3. Let v be an arbitrary vertex in $H_1 \cap H_2$, and v_1, v_2 any vertices in $H_1 \setminus H_2, H_2 \setminus H_1$, respectively.

Suppose $(v_1, v) \in E(G)$. Since H_2 is a guess component, v_1 is adjacent to all vertices in H_2. Since H_1 is a guess component, each vertex in $H_2 \setminus H_1$ is adjacent to each vertex in H_1. Since H_2 is a guess component, each vertex in $H_1 \setminus H_2$ is adjacent to each vertex in H_2. Similarly, if $(v_1, v) \notin E(G)$, then there are no edges between $H_2 \setminus H_1$ and H_1, as well as between $H_1 \setminus H_2$ and H_2.

Using the fact that H_1 is a guess component, and the fact that H_2 is a guess component, we obtain

$$N(v_1) \setminus (H_1 \cup H_2) = N(v) \setminus (H_1 \cup H_2) = N(v_2) \setminus (H_1 \cup H_2).$$

Hence $H_1 \cup H_2$ is a guess component.

According to parts (1) and (2), for each vertex v' in $H_1 \cap H_2$

$$N(v) \setminus (H_1 \cap H_2) = [N(v) \cap (H_1 \Delta H_2)] \cup [N(v) \setminus (H_1 \cup H_2)] = [N(v') \cap (H_1 \Delta H_2)] \cup [N(v') \setminus (H_1 \cup H_2)] = N(v') \setminus (H_1 \cap H_2). \tag{3.1}$$

According to part (3), for each vertex v'_1 in $H_1 \setminus H_2$

$$N(v_1) \setminus (H_1 \setminus H_2) = [N(v_1) \cap (H_1 \cap H_2)] \cup [N(v_1) \setminus H_1] = [N(v'_1) \cap (H_1 \cap H_2)] \cup [N(v'_1) \setminus H_1] = N(v'_1) \setminus (H_1 \setminus H_2). \tag{3.2}$$

According to parts (1) and (2), for each pair of vertices v_3, v'_3 in $H_1 \Delta H_2$

$$N(v_3) \setminus (H_1 \Delta H_2) = [N(v_3) \cap (H_1 \cap H_2)] \cup [N(v_3) \setminus (H_1 \cup H_2)] = [N(v'_3) \cap (H_1 \cap H_2)] \cup [N(v'_3) \setminus (H_1 \cup H_2)] = N(v'_3) \setminus (H_1 \Delta H_2). \tag{3.3}$$

Hence $H_1 \cap H_2, H_1 \setminus H_2$ and $H_1 \Delta H_2$ are also guess components. □

A guess component is *trivial* if it is either V or a singleton. A guess component $H \neq V$ is *maximal* if the only guess component containing it is V. Similarly, a guess component H with $|H| \geqslant 2$ is *minimal* if every component contained in it is trivial.

Lemma 3.6. *If G is not connected then the maximal guess components in G are the complements of the connected components, namely the sets*

$$\bigcup_{i\neq j} H_i, \quad 1 \leqslant j \leqslant m, \tag{3.4}$$

where $H_1, \ldots, H_m$ (with $m \geqslant 2$) are the connected components of G.

Proof. Obviously, the sets in (3.4) are guess components. Assume to the contrary that there exists a maximal guess component $H = H'_1 \cup, \ldots, \cup H'_m$ such that $\emptyset \neq H'_i \subseteq H_i$ for $1 \leqslant i \leqslant m$. Since $H \neq V$ and without loss of generality we may assume $H'_m \subsetneq H_m$. Since H_m is connected, there exists a vertex $v \in H_m \setminus H'_m$ that is adjacent to H'_m. On the other hand, v is not adjacent to any vertex in H_1. This contradicts the fact that H is a guess component. Thus the sets in (3.4) are the only maximal guess components. □

Lemma 3.7. *For any graph G, exactly one of the following conditions holds:*

1. *No two maximal guess components intersect.*
2. *Every two maximal guess components intersect and their union is V.*

Proof. If G contains two intersecting maximal guess components H_1 and H_2, then, by Lemma 3.3, $H_1 \cup H_2$ is also a guess component. Since H_1 is maximal, $H_1 \cup H_2 = V$. Any other maximal guess component intersects H_1, since otherwise it would be contained in H_2. By the same reasoning, for any other maximal components H_3, H_4 we get $H_1 \cup H_3 = H_1 \cup H_4 = V$. Hence H_3 intersects H_4. □

For a graph $G = (V, E)$, denote by $\overline{G} = (V, V \times V \setminus E(G))$, the complement of G. It is easy to see that a set $H \subseteq V$ is a guess component of G if and only if it is a guess component in $\overline{G}$.

Lemma 3.8. *G contains intersecting maximal guess components if and only if either G or $\overline{G}$ consists of at least 3 connected components.*

Proof. The "if" part follows from Lemma 3.6 and the observation that G and $\overline{G}$ have the same guess components. Now suppose G contains two intersecting maximal guess components H_1, H_2. By Lemma 3.3 the set $[H_1 \times (H_2 \setminus H_1)] \cup [H_2 \times (H_1 \setminus H_2)]$ is either contained in $E(G)$ or disjoint from it. This means that either in G or in $\overline{G}$, there are no edges between the sets $H_1 \setminus H_2, H_2 \setminus H_1$ and $H_1 \cap H_2$, which means that G or $\overline{G}$ consists of at least three connected components. This concludes the "only if" part. □

Corollary 3.9. *Let $\{H_1, \ldots, H_k\}$ be the set of maximal guess components. If V contains two intersecting maximal guess components then:*

1. $H_j \supseteq V \setminus \bigcap_{i\neq j} H_i, \qquad 1 \leqslant j \leqslant k.$
2. $k \geqslant 3$.
3. Every vertex belongs to at least two maximal guess components.

Corollary 3.10. *There are at most $|V|$ maximal guess components in V.*

Proof. If no two maximal guess components intersect the corollary is trivial. Otherwise, we conclude it from Lemmas 3.6 and 3.8. □

For a set of vertices $H \subseteq V$ and a pair of vertices $u, v \in H$, denote by $H(u, v)$ the largest guess component in the subgraph induced by H that includes v but not u. (Note that this guess component is not necessarily maximal according to the definition above. For example, the largest guess component including v_6 but not v_7 in the graph depicted in Fig. 3 consists of v_6 only, and is properly contained in the maximal component $\{v_6, v_7\}$). By Lemma 3.3, $H(u, v)$ is well defined.

Denote by v_i the cell uncovered at the ith iteration and by v^* the mined cell.

Lemma 3.11. *If, for some $i_1 > 1$, we have $Safe_{i_1} = \emptyset$ and $Safe_i \neq \emptyset$ for every $1 < i < i_1$, then $Covered_{i_1} = V(v_1, v^*)$.*

Proof. By Corollary 3.2, $Covered_{i_1}$ is a guess component. We prove by induction on i that for $i < i_1, Safe_i \cap V(v_1, v^*) = \emptyset$ for $i < i_1$. For $i = 1$, the claim is trivial. Suppose the lemma holds for some $1 \leqslant i < i_1 - 1$. Since either $i = 1$ or $Safe_i \neq \emptyset$, we get by the induction hypothesis that $v_i \notin V(v_1, v^*)$. If $label(v_i) = 0$, then v_i is not a neighbor of the mined cell, and hence it is not a neighbor of any cell in $V(v_1, v^*)$. Similarly, if $label(v_i) = 1$, then v_i is a neighbor of the mined cell, and hence it is a neighbor of all cells in $V(v_1, v^*)$. Hence $Safe_{i+1}$ does not intersect $V(v_1, v^*)$. This means that $V(v_1, v^*) \subseteq Covered_{i_1}$. Since $Covered_{i_1}$ is a guess component that includes the mined cell but not v_1, we get $V(v_1, v^*) = Covered_{i_1}$. □

The following algorithm computes the largest guess component required to include some specific vertex but not another.

Algorithm 3. Finding the largest guess component including v but not u

```
LargestComponent (Graph G, Vertex u, Vertex v)
  Set Safe ← {u}
  Set Covered ← V
  while Safe ≠ ∅
    v' ← an arbitrary vertex in Safe
    Safe ← Safe \ {v'}
    Covered ← Covered \ {v'}
    if v ∉ N(v')
      Safe ← (Safe ∪ N(v')) ∩ Covered
    else
      Safe ← (Safe ∪ (V \ N(v'))) ∩ Covered
  return Covered
```

Lemma 3.12. Algorithm 3 *finds the largest guess component including v but not u in* $O(|V|^2)$ *time.*

Proof. Algorithm 3 is a simulation of Algorithm 2, where the cell v_1 uncovered at the first iteration is u and v is the mined cell. The simulation is executed until the first time $Safe = \emptyset$, and by Lemma 3.2 we get the result. Each iteration in the main loop of Algorithm 2 takes $O(|V|)$ time. There are at most $|V|$ iterations until the next time in which $Safe = \emptyset$. Hence the total runtime is $O(|V|^2)$. □

Theorem 3.13. *It is possible to find all maximal guess components in V in* $O(|V|^3)$ *time and* $O(|V|^2)$ *memory.*

Proof. One can run Algorithm 3 for an arbitrary vertex v and for every vertex $u \neq v$. At least one of these components is maximal. Thus in time $O(|V|^3)$ we can find all maximal guess components including v. Denote one of these components by H. Distinguish between two cases:

Case (i) H is the only maximal component including v.
By Corollary 3.9, no two maximal guess components intersect. Choose an arbitrary vertex $u' \in H$ and run Algorithm 3 with u' and every other vertex $v' \neq u'$ in $V \setminus H$ in place of u and v, respectively. Since any other maximal guess component does not include u', we are guaranteed to find all maximal guess components. This takes $O(|V|^3)$ time.

Case (ii) H is not the only maximal component including v. By Lemma 3.7, every two maximal guess components intersect. Choose an arbitrary vertex $v'' \in V \setminus H$ and run Algorithm 3 with v'' and every other vertex $u'' \in V$. Since any other maximal guess component includes v'' we are guaranteed to find all maximal guess components. This takes $O(|V|^3)$. □

4. Proof of the main results

We begin by proving the lemmas in Section 2.

Proof of Lemma 2.1. We proceed by induction. For $i = 1$, the lemma is trivial. Suppose the lemma holds for $i = m$. In the mth iteration, a cell v is uncovered, and *Safe* is updated according to $label(v)$. If $label(v) = 0$, then all of v's neighbors are free. In this case, every assignment that is legal at time m, and in which some non-neighbor of v is the mined cell, remains legal at time $m + 1$. If $label(v) = 1$, then the only mine is in a neighbor of v, and hence all of v's non-neighbors are free. In this case, every assignment that is legal at time m, and in which some neighbor of v is the mined cell, remains legal at time $m + 1$. Hence, by the induction hypothesis, $Safe_{m+1}$ is composed exactly of all covered cells that are free in every legal assignment of G at time $m + 1$. □

Proof of Lemma 2.2. Denote $k = |Covered_i|$. By Lemma 2.1, for every two cells $v_1, v_2 \in Covered_i$ and every $v' \in (V \setminus Covered_i) \cap N(v_1)$, we must have $label(v') = 1$ and also $v' \in N(v_2)$. Indeed, otherwise, the set *Safe* would have included either v_1 or v_2. Thus, for any possible location of the mine, the labels in the uncovered cells would all be the same. Hence the a priori probability of the event E, whereby the uncovered cells contain the labels that were eventually uncovered, is $\frac{k}{n}$. Hence

$$P(label(v_1) = *|E) = \frac{\frac{1}{n}}{\frac{k}{n}} = \frac{1}{k}. \qquad \square$$

Proof of Corollary 2.3. Denote by P_i^* the probability in question. Since G includes the uncovered cells, every strategy for minesweeping $Covered_i$ can be implemented in order to continue minesweeping G from the ith stage on. In particular $P_i^* \geqslant P^*(Covered_i)$. Now let s be any strategy for minesweeping the graph G, given that $V \setminus Covered_i$ is free and given the values in all cells of this set. By Lemmas 2.1 and 2.2, when given only the subgraph $Covered_i$, we can reconstruct G with all the above information regarding $V \setminus Covered_i$. Hence we can implement for $Covered_i$ a strategy with the same winning probability as s, and in particular $P^*(Covered_i) \geqslant P_i^*$. $\square$

We continue by proving several lemmas concerning optimal strategies.

Lemma 4.1. $P^*(\overline{G}) = P^*(G)$.

Proof. For every vertex v in V, the uncovering of v gives exactly the same information for G and for $\overline{G}$. Hence every strategy for minesweeping G can be implemented in order to continue minesweeping $\overline{G}$. In particular $P^*(\overline{G}) \geqslant P^*(G)$. Similarly we get $P^*(\overline{G}) \leqslant P^*(G)$. $\square$

We recall that we may assume that the method OptimalGuess is always applied to an MS graph with no uncovered vertices.

Lemma 4.2. *If $E = V \times V$ then $P(G) = \frac{1}{n}$.*

Lemma 4.3. *If G includes a vertex v with no neighbors, then there exists an optimal strategy s^* for which $v_1 \neq v$.*

Proof. Any strategy s' that uncovers v at this stage either leads to an immediate loss (with probability $\frac{1}{|V|}$) or yields no information from this move (beyond the fact that v is free). A strategy s'' that postpones the uncovering of v has the same probability for an immediate loss, but may obtain some information and thereby raise the winning probability. Hence $P_{s''} \geqslant P_{s'}$. $\square$

Lemma 4.4. *Let H be the set of all non-isolated vertices in G (i.e., vertices of valency at least 1). Suppose $\emptyset \neq H \neq V$. Then $P^*(G) = \frac{|H|}{n} \cdot P^*(H) + \frac{1}{n}$.*

Proof. From Lemma 4.3 we get that there exists an optimal strategy that uncovers all of H (except for the mined vertex if it is there) before uncovering any vertex of $V \setminus H$. If the mined cell is in H and is covered, then since all of its neighbors have label 1, all vertices in $V \setminus H$ can be safely uncovered and we win. Thus the winning probability in this case is $P^*(H)$. Otherwise, we eventually arrive at the situation whereby $Covered = V \setminus H$. The probability of guessing the mined cell in this case is $\frac{1}{|V \setminus H|}$. Altogether:

$$P^*(G) = \frac{|H|}{n} \cdot P^*(H) + \frac{|V \setminus H|}{n} \cdot \frac{1}{|V \setminus H|} = \frac{|H|}{n} \cdot P^*(H) + \frac{1}{n}. \qquad \square \tag{4.1}$$

Corollary 4.5. *In the setup of Lemma 4.4, if $|V \setminus H| > 1$, then there exists an optimal strategy for which $v_1 \in V \setminus H$.*

Proof. By Lemma 4.3, the winning probability of the strategy uncovering a vertex from $V \setminus H$ is

$$\frac{n-1}{n} \cdot \left(\frac{|H|}{n-1} \cdot P^*(H) + \frac{1}{n-1} \right) = \frac{|H|}{n} \cdot P^*(H) + \frac{1}{n} = P^*(G). \qquad \square$$

By Lemma 4.3 we assume from now on, without loss of generality, that there are no isolated vertices in G. By Lemma 4.1 we may symmetrically assume that there are no vertices of full degree.

Lemma 4.6. *Let V be a disjoint union of subsets $H_1, \ldots, H_m$, with no edges between distinct H_i's. Assume that the subgraphs induced by the H_i's are connected. Denote $k_i = |H_i|$ and $P_i = P^*(H_i)$ for $i = 1, \ldots, m$. Then*

$$P^*(G) = \sum_{i=1}^{m} \frac{k_i}{n} \cdot P_i. \tag{4.2}$$

Proof. We prove the lemma by induction on m. For $m = 1$ the lemma is trivial. Due to the symmetry of the right-hand side of (4.2), we may assume without loss of generality that the first guess is in H_1. If the mined cell is not in H_1, then, by Lemma 3.11, at the time of the second guess $Covered = V \setminus H_1$, and the winning probability is therefore $P^*(V \setminus H_1)$. By the induction

hypothesis $P^*(V \setminus H_1) = \sum_{i=2}^{m} \frac{k_i}{n-k_1} \cdot P_i$. Otherwise, since not all vertices in H_1 are uncovered before the second guess, it can be deduced that all vertices in $V \setminus H_1$ are free. Hence, by Lemma 2.1, all vertices in $V \setminus H_1$ will be uncovered before the second guess, so that the winning probability is P_1. Since the a priori probabilities for the mine to reside in H_1 and in $V \setminus H_1$ are $\frac{k_1}{n}$ and $\frac{n-k_1}{n}$, respectively, the winning probability is as on the right-hand side of (4.2). □

Lemma 4.7. *Suppose H_1, H_2 are two maximal guess components with $H_1 \cap H_2 \neq \emptyset$. Denote $k_1 = |H_1|, k_{2\setminus 1} = |H_2 \setminus H_1|, P_1 = P^*(H_1)$ and $P_{2\setminus 1} = P^*(H_2 \setminus H_1)$. Then:*

$$P^*(G) = \frac{k_1}{n} \cdot P_1 + \frac{k_{2\setminus 1}}{n} \cdot P_{2\setminus 1}.$$

Proof. By the proof of Lemma 3.7 we have $H_1 \cup H_2 = V$. By Lemmas 3.3 and 4.1, we may assume without loss of generality that there are no edges between H_1 and $H_2 \setminus H_1$. According to our assumption, whereby there are no isolated vertices, $H_2 \setminus H_1$ is a non-trivial guess component. We get $P^*(G)$ by Lemma 4.6. □

Note that the lemma does not hold in general without making the assumption made above, whereby there are no isolated vertices. Indeed, for a graph G without any edges we have $P^*(G) < 1$, although the winning probability for each connected component in this case is 1.

Lemma 4.8. *Let V be a disjoint union of m non-trivial maximal guess components $H_1, \ldots, H_m$ and l trivial maximal guess components $\{u_1\}, \ldots, \{u_l\}$. Denote $k_i = |H_i|$ and $P_i = P^*(H_i)$ for $i = 1, \ldots, m$. Then $m \leqslant 1$ then*

$$P^*(G) = \begin{cases} \frac{l-1}{n}, & m = 0, \\ \frac{l}{n} + \sum_{i=1}^{m} \frac{k_i}{n} \cdot P_i, & m > 0 \end{cases}.$$

Proof. If the first guess is some u_i then with probability $\frac{1}{n}$ the game is immediately lost. If the game is not immediately lost, then by Lemma 3.11 all cells but those in the maximal guess component including the mined cell are uncovered before we make the second guess. Since the a priori probability for the mine to be in each H_i is $\frac{k_i}{n}$, the winning probability in this case is:

$$\frac{l-1}{n} \cdot 1 + \sum_{i=1}^{m} \frac{k_i}{n} \cdot P_i.$$

If the first guess is in some H_i then the calculation is similar, unless the mined cell v^* is in H_i. In the latter case, by Lemma 3.11, all cells but these in $H_i(v_1, v^*)$ are uncovered. Thus the winning probability in this case is

$$\frac{l}{n} + \sum_{i=1}^{m} \frac{k_i}{n} \cdot P_i. \qquad \square$$

Lemma 4.9. *Let H be a non-trivial maximal guess component. Then there exists an optimal strategy for which $v_1 \in H$.*

Proof. If H intersects another maximal guess component, then by the proof of Lemma 4.7 there exists an optimal strategy for which $v_1 \in H$. Otherwise, V is a disjoint union of maximal guess components, and by the proof of Lemma 4.8 there exists an optimal strategy for which $v_1 \in H$. □

Algorithm 4 is a recursive algorithm that returns an optimal guess. It simulates the (continuation of the) game for any possible location of the mined cell. If it finds that, for some initial guess, no additional guesses are required, then the returned value is that guess. Otherwise, it finds an optimal guess in a non-trivial guess component and returns it.

Algorithm 4. An algorithm finding an optimal guess

```
OptimalGuess (Graph G)
  u ← an arbitrary cell in G
  for each cell v ≠ u
    H ← LargestComponent (G, u, v)
    if |H| > 1
      return OptimalGuess (G|H)
  return u
```

Lemma 4.10. *Algorithm 4 works in $O(|V|^3)$ time.*

Proof. By Lemma 3.12 every call to LargestComponent takes $O(|V|^2)$ time. If a call to LargestComponent (G, u, v) returns a trivial guess component, then v will not appear again as a third parameter in any future call to LargestComponent (whether in the current for loop or via some recursive call to OptimalGuess). If it does return some component H with $|H| > 1$, then we get rid of at least one vertex, namely u. Therefore there are at most $|V| - 1$ invocations of LargestComponent, so that the total time of the algorithm is at most $O(|V|^3)$. □

Lemma 4.11. *Algorithm 4 returns an optimal guess.*

Proof. If all guess components not containing u are trivial, then by uncovering u we have a winning probability of $1 - \frac{1}{n}$, namely we win unless u is the mined vertex. Since this probability cannot be surpassed, the choice of u in this case is optimal. If some non-trivial guess component is found, then the returned value is an optimal choice from a maximal guess component. By Lemma 4.9 this is an optimal guess. □

Proof of Theorem 2.4. The first claim is the contents of Proposition 4.11. By Lemma 4.10, each call of Algorithm 4 takes $O(|V|^3)$ time. Since Algorithm 4 is called $O(|V|)$ times, we get the claimed runtime. □

For the next proof, note that in a tree the only non-trivial guess components are sets of leaves with the same neighbor.

Proof of Proposition 2.5. By Lemma 3.8, since T is connected, V is a disjoint union of maximal guess components. The proposition now follows from Lemma 4.8. □

Proof of Proposition 2.6. If $n_1 = 1$ then G is a tree. In this case, if $n_2 = 3$ then G contains one non-trivial maximal guess component and one trivial maximal guess component. If $n_2 \neq 3$, then G contains only trivial maximal guess components. By Proposition 2.5, in both cases $P(G) = \frac{n-1}{n}$.

If $n_1 = n_2 = 2$, then G is a clique with 4 vertices and by Lemma 4.2 we have $P(G) = \frac{1}{4}$.

If $n_1 = 2$ and $n_2 > 2$, then the maximal guess components are pairs of vertices lying in the same column. By Lemma 4.8 we get

$$P(G) = \sum_{i=1}^{n_2} \frac{2}{2n_2}\frac{1}{2} = \frac{1}{2}.$$

If $n_1 = n_2 = 3$, denote by $c(G)$ the central cell of G. The only non-trivial guess component is $H = V \setminus c(G)$. Thus the first guess will be one of the cells in H and by the time of the second guess *Covered* is a singleton.

If $n_1 \geqslant 3$ and $n_2 > 3$, then all maximal guess components are trivial. In fact, assume to the contrary that H is a maximal non-trivial guess component. Denote by v a vertex outside which is adjacent to some vertex in H. Since H is a guess component, all of its vertices are adjacent to v and thus H is contained in some 3×3 grid G'. On the other hand, since H is a guess component of G, it is a guess component of G' as well. Thus $H = V(G') \setminus \{c(G')\}$. Since $n_2 > 3$ there exists a cell outside of H that is connected only to a part of H, in contradiction to H being a guess component. □

Proof of Proposition 2.7. By Lemma 4.1 we may assume $p \geqslant \frac{1}{2}$. For a set of vertices $V_1 \subseteq V$, if V_1 is a guess component, then there exist two disjoint sets $V_2, V_3 \subseteq V$ for which $V_2 \cup V_3 = V \setminus V_1$ and $V_1 \times V_2 \subseteq E(G)$ and $V_1 \times V_3 \subseteq E(\overline{G})$. This means that, if $|V_1| = k$, then the probability of V_1 being a guess component is

$$\sum_{l=0}^{n-k} \binom{n-k}{l} p^{kl}(1-p)^{k(n-k-l)} = \left(p^k + (1-p)^k\right)^{n-k}.$$

Since $p \geqslant \frac{1}{2}$,

$$p^k + (1-p)^k \leqslant p \cdot p^{k-1} + (1-p) \cdot p^{k-1} = p^{k-1}.$$

Thus, the probability for the existence of a guess component of size k is at most $\binom{n}{k} p^{(k-1)(n-k)}$. The probability for the existence of a non-trivial guess component is therefore at most

$$\sum_{k=2}^{n-1} \binom{n}{k} p^{(k-1)(n-k)} = 2\sum_{k=2}^{k_0} \binom{n}{k} p^{(k-1)(n-k)} + \sum_{k=k_0+1}^{n-k_0} \binom{n}{k} p^{(k-1)(n-k)} \leqslant 2n^{k_0}p^{n-2} + 2^n p^{k_0(n-k_0-1)} \tag{4.3}$$

for any constant $k_0 \geqslant 2$ and large enough n. In particular, if we put

$$k_0 = \left\lceil 1 + 2\log_p \frac{1}{2} \right\rceil,$$

so that

$$k_0(n - k_0 - 1) = nk_0 - k_0^2 - k_0 > n(k_0 - 1) = 2n\log_p \frac{1}{2},$$

we obtain

$$2n^{k_0}p^{n-2} + 2^n p^{k_0(n-k_0-1)} \leqslant 2n^{k_0}p^{n-2} + 2^n p^{2n\log_p \frac{1}{2}} = 2n^{k_0}p^{n-2} + 2^n \left(\frac{1}{2}\right)^{2n} = 2n^{k_0}p^{n-2} + \left(\frac{1}{2}\right)^n. \tag{4.4}$$

The last expression tends to zero as n tends to infinity. □

5. Conclusion

To conclude, we present an optimal strategy for Minesweeper graphs containing a single mine. Our method involves the definition of guess components, which have interesting properties. We present a method to calculate the winning probability for any given graph. We also discuss the winning probability for trees and random graphs. These results may be used in order to develop a more successful strategy for the original game.

References

[1] J.D. Ramsdell, Programmer's Minesweeper. <http://www.ccs.neu.edu/home/ramsdell/pgms/>.
[2] C. Studholme, Minesweeper as a Constraint Satisfaction Problem. <http://www.cs.utoronto.ca/~cvs/minesweeper/>.
[3] J. Rhee, Evolving Strategies for the Minesweeper Game using Genetic Programming, Genetic Algorithms and Genetic Programming at Stanford 6 (2000) 312–318, Stanford Bookstore, California.
[4] C. Quartetti, Evolving a Program to Play the Game Minesweeper, Genetic Algorithms and Genetic Programming at Stanford 6 (2000) 137–146, Stanford Bookstore, California.
[5] P. Jordan, Evolving minesweeper strategies using genetic algorithms (CS 472 term paper).
[6] L. Castillo, S. Wrobel, Learning minesweeper with multirelational learning, Proc. 18th Int. Joint Conf. Artif. Intell. (2003) 533–538.
[7] J. Bellaiche, Minesweeper with reinforcement learning neural networks (term paper), 1998.
[8] A. Adamatzky, How cellular automaton plays Minesweeper, Appl. Math. Comput. 85 (2–3) (1997). 127–123.
[9] S. Golan, Minesweeper on graphs, Appl. Math. Comput. 217 (14) (2011) 6616–6623.
[10] O. German, E. Lakshtanov, Minesweeper and spectrum of discrete Laplacians, Appl. Anal. 89 (12) (2010) 1907–1916.
[11] R. Kaye, Infinite versions of minesweeper are Turing complete. <http://for.mat.bham.ac.uk/R.W.Kaye/minesw/infmsw.pdf>.
[12] E. Mossel, The Minesweeper game: percolation and complexity, Combin. Prob. Comput. 11 (5) (2002) 487–499. <http://heinrichmartin.com/mw/moreMWG.pdf>.
[13] R. Kaye, Minesweeper is NP-complete, Math. Intellig. 22 (2000) 9–15.
[14] M. Heinrich, More properties for NP-complete Minesweeper graphs.